Chap 4

Syntax Directed Translation & it's appl

Grammar + Semantic Rule = SDT

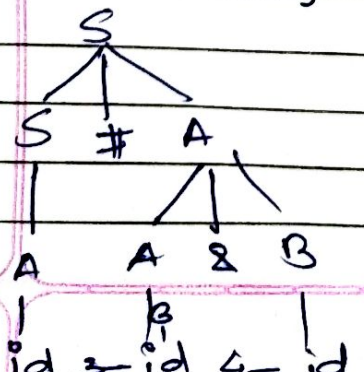
Application

1) Expecting Arithmatic Expression
2) conversion from infix to postfix
3) conversion from infix to prefix
4) conversion from binary to decimal
5) Counting no. of reductions
6) Creating Syntax tree
7) Generating intermediate code
8) Type checking
9) Storing type info. into Symbol table

Grammar	rule
$S \rightarrow S \# A \mid A$	{ S.val = S.val * A.val; }
$A \rightarrow A \& B \mid B$	{ A.val = A.val + B.val ; }
$B \rightarrow id$	{ B.val = id.val ; }

given string 5 # 3 & 4

SA → Annotated
Symbatic analysis



5 * 3 + 4
= 19

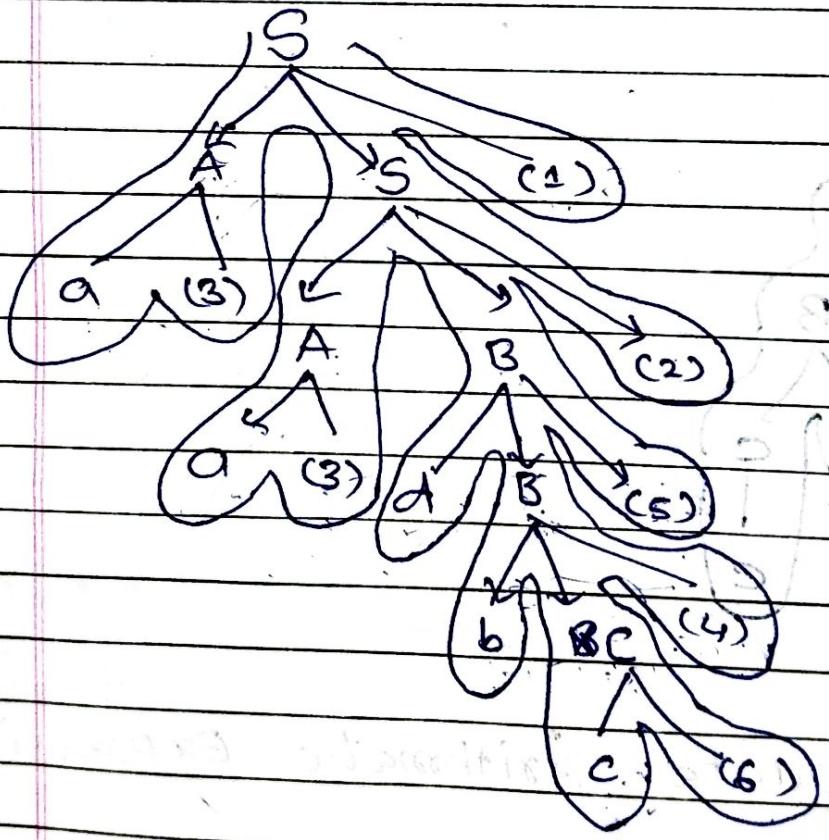
Syntax Directed Translation – Evaluation

$S \rightarrow AS$ { printf(1) }	1
$S \rightarrow AB$ { printf(2) }	2
$A \rightarrow a$ { printf(3) }	3
$B \rightarrow bC$ { Pf (4) }	4
$B \rightarrow dB$ { Pf (5) }	5
$C \rightarrow c$ { Pf (6) }	6

I/P → aadbc

O/P → 3 3 6 4 5 2 1

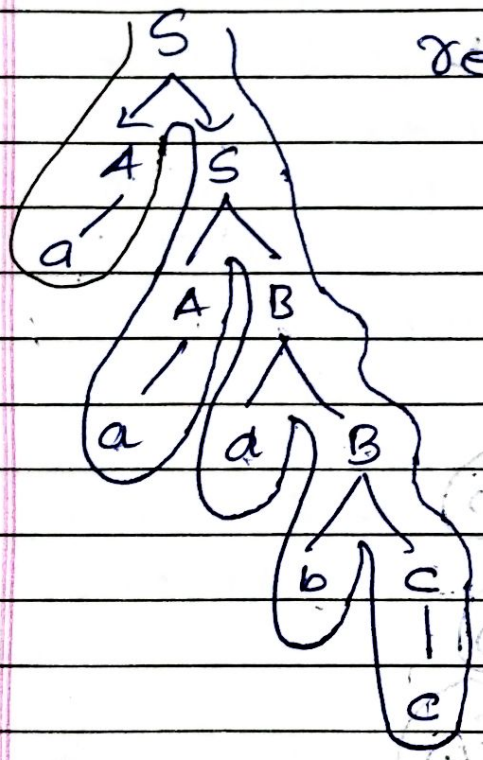
Top Down Approach



Bottom up

S → As {printf(1)} (1)
S → AB {Pf(2)} (2)
A → a {Pf(3)} (3)
B → bC {Pf(4)} (4)
B → dB {Pf(5)} (5)
C → c {Pf(6)} (6)

I/p – aadbc
O/p – 3364521



reduction based

* How to Evaluate: Arithmatic Expression

Types of SDT

S-Attributed SDT	L-Attributed
- Based on Synthesized attribute	- Based on both synthesized & a inherited attribute (Parent, *Left sibblings)
- Use bottom up parsing	- Top down Parsing
- Semantic Rules always written at Rightmost position in RHS.	- Semantic Rules anywhere in RHS

Inherit

$A \to LM$ & $L.i = l(A.i)$; Synthesis

$M.i = m(L.s)$; $A.s = f(M.s)$

$A \to QR$ & $R.i = r(A.i)$; $Q.i = q(R.s)$; Inherit X

$A.s = f(Q.s)$;

Synthesis

$A \to XYZW$

→ Value taken from child to parent is called Synthesised

→ Sibbling take value is called inherited.

Not L-attributed not even S-Attribute.

Synthesised

Gate $A \to PQ$ and $A \to XY$

2020 { Rule 1 : $P.i = A.i + 2$, $Q.i = P.i + A.i$ and $A.S = P.S + Q.S$ } L-attributed

{ Rule 2 : $X.i = A.i + Y.S$ and $Y.i = X.S + A.i$ } Circular

which of the following is true?

A) Both Rules are L-Attributed
B) only R_1 is L-attribute
c) only R_2 is L-Attribute
d) Neither R_1 nor R_2 L-attributed

Ans — (B) Only R_1 is L-attribute

S-Attributed

1. A SDD that uses only Synthesized attribute is called S-Attributed SDD.

Ex. $A \longrightarrow BCD$

$A.i \Rightarrow B.i$

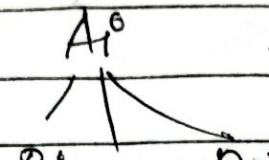
$A.i = C.i$

$A.i = D.i$

2. Semantic actions are always placed at right end of the production. (POSTFIX SDD)

3. Attribute are evaluated with

Value of A_i - can calculate from B, C, D i.e children.
S - Synthesized



→ Value of A_i - can calculated from child - Synthesized or from child to Parent & Sibling to Siblings is called Synthesized.

L-Attribute

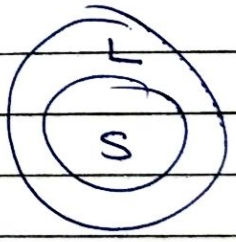
A SDD that uses both synthesized & Inherited attributes is called as L-attributed SDD ~~that~~ but each inherited attribute is restricted to inherit from parent or Left Siblings.

Ex. $A \rightarrow BCD$

$C_i^\circ = A_i^\circ$

$C_i^\circ = B_i^\circ$

$C_i^\circ = D_i^\circ$



- Semantic action are placed anywhere on RHS of the productions.

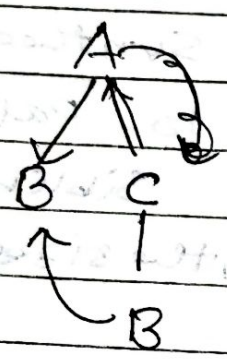
- Attribute are evaluated by traversing parse tree depth first, Left to right

In In Syn

1) $A \to Bc$ { $B.i = A.i$, $C.i = B.i$ } $\to A.i = C.i$ }

2) $A \to QR$ { $R.i = A.i$, $Q.i = R.i$ } $A.i = Q.i$ }
In Syn

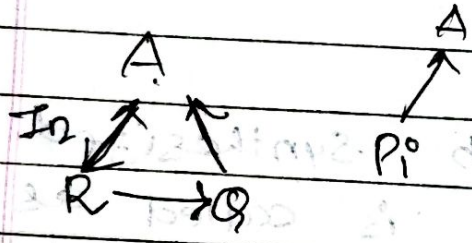
3) $A \to PQ$ { $A.i = P.i$, $A.i = Q.i$ }
Syn Syn



1) L - attribute

2) Neither S, L attributed

3) Both S & L



Syntax Directed def^{n} (SDD)

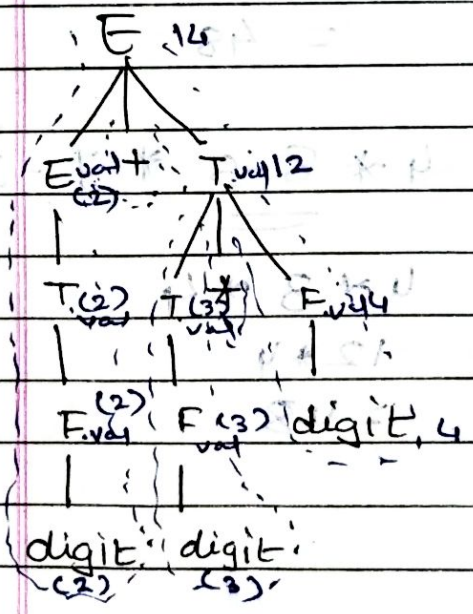
SDD = Grammar + Semantic rule

Production	Semantic rule
$E \rightarrow E+T$	E.val = E.val + T.val
$E \rightarrow T$	E.val = T.val
$T \rightarrow T*F$	T.val = T.val * F.val
$T \rightarrow F$	T.val = F.val
$F \rightarrow$ digit	F.val = digit val

1. Attributes are associated with grammar Symbol & Semantic rule are associated with productions.

2. Attributes may be number, strings, refrence, data types etc

2 + 3 * 4 (TOP to down - left to right



- 2 + 3 * 4 = 14

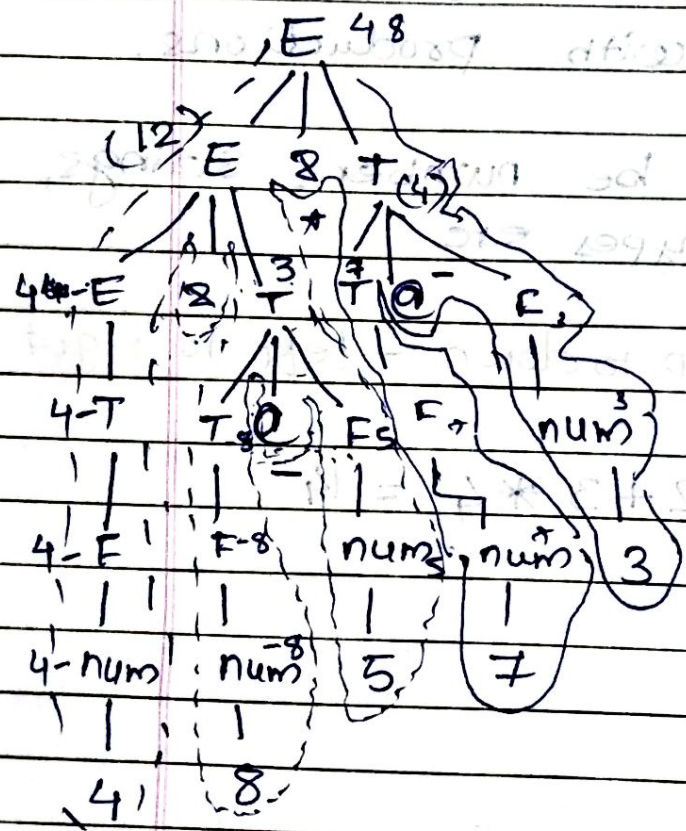
$E \rightarrow E \,@\, T$ { $E.val = E.val * T.val$ } (1)
$\quad | \; T$ { $E.val = T.val$ } (2)

$T \rightarrow T @ F$ { $T.val = T.val - F.val$ } (3)
$\quad | \; F$ { $T.val = F.val$ } (4)

$F \rightarrow num$ { $F.val = num$ } (5)

I/P – 4 & 8 @ 5 & 7 @ 3

Bottom up O/P ⇒ 48



4 * 8 - 5 * 7 - 3

4 * 8 - 5 * 4

4 * 3 * 4

⇒ 12 * 4

= 48

4 * 8 - 5 * 7 - 3

4 * 3 * 4

12 * 4

= 48

SDD - Specifies the values of attributes by associating Semantic rules with the productions.

SDT - embeds programs fragments (also called Semantic actions) within production bodies.

	SDD	SDT
1.	Attribute grammar	1) Translation scheme
2.	Production with Semantic rules	2) Production with Semantic actions
3.	$E \rightarrow E_1 + T$ $\{E.val = E.val + T.val\}$	3) $E \rightarrow E_1 + T$ $\{$ Prints '+' $\}$
4.	more readable	4. more efficient
5.	Useful for Specification	5. useful for implementation
6.	The grammar either Specifies what calculations have to done	It can be used to implement two classes of SDD. S-attributed SDD L-attributed SDD
7.	Order of	6. grammar writes Specification both the Calculation that have to be done & at

time they must be di

7. Order of Evaluation is Unspecified, it must be fixed by parser (bottom up or ~~to~~ top-down)

8. Attributes of node may be concerned with data type, numerical value, Symbol identification, memory address.

8. Action may perfo arbitary computation such as appending o/p